Universität Bremen

Institute for Artificial Intelligence

Faculty 03

Mathematics &

Computer Science

Robot Programming with ROS

3. Robots and Communication

Arthur Niedźwiecki, Stefan Eirich

26th Oct. 2023

Overview

Overview

Industrial Robots

Logistics

Image courtesy: BIBA

Automotive

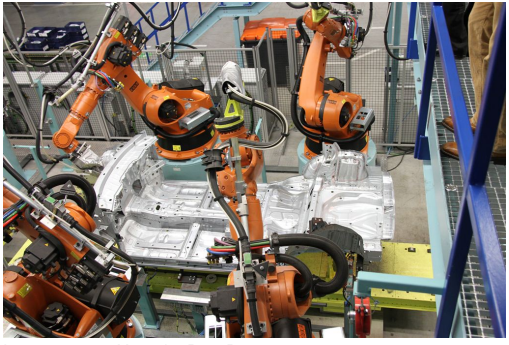


Image courtesy: Mercedes Benz Bremen

- Extremely heavy, precise and dangerous, not really smart
- Mostly no sensors, only high-precision motor encoders
- Programmable through PLCs (using block diagrams or Pascal / Basic like languages)

Industrial Light-weight Robots

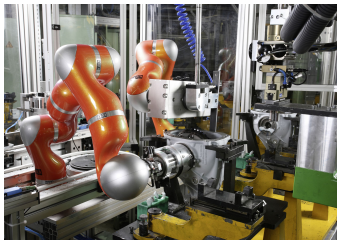
Production:

Copyright: Universal Robots

Medicine:

Copyright: Intuitive Surgical

Automotive:



Copyright: KUKA Roboter GmbH

- Very precise, moderately dangerous, somewhat smart
- High-precision motor encoders, sometimes force sensors, cameras
- Native programming and simulation tools (C++, Java, Python, GUIs)

Service Robots

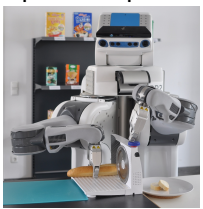
Autonomous aircrafts

Courtesy DJI

Mobile platforms

Courtesy NASA/JPL-Caltech

Manipulation platforms



Humanoids

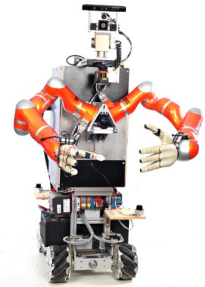
- Usually not very precise
- Not really dangerous
- Usually cognition-enabled
- Equipped with lots of sensors
- Usually running Linux

Service Robots with Light-weight Arms

DLR Justin

Courtesy of DLR

TUM Rosie



- Moderately precise and dangerous
- Cognition-enabled
- Equipped with lots of sensors
- Usually running a combination of a real-time and non real-time OS.

Overview

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems

Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems
  - Distributed processing over multiple computers / robots

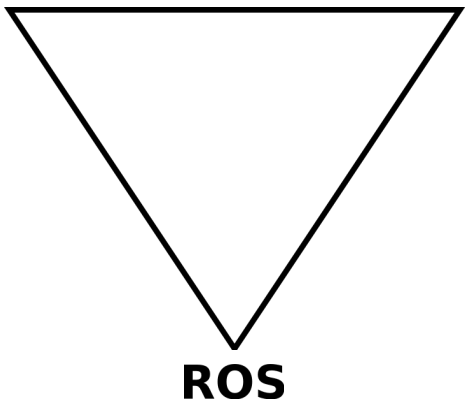
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with. Requirements:
  - Support for different programming languages
  - Different operating systems
  - Distributed processing over multiple computers / robots
  - Easy software sharing mechanisms

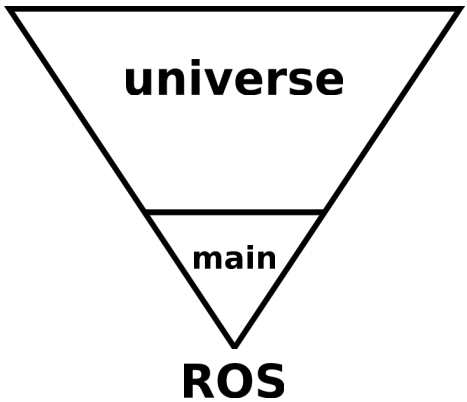
Robot Operating System

At 2007 Willow Garage, a company founded by an early Google employee Scott Hassan at 2006 in the Silicon Valley, starts working on their Personal Robotics project and ROS.

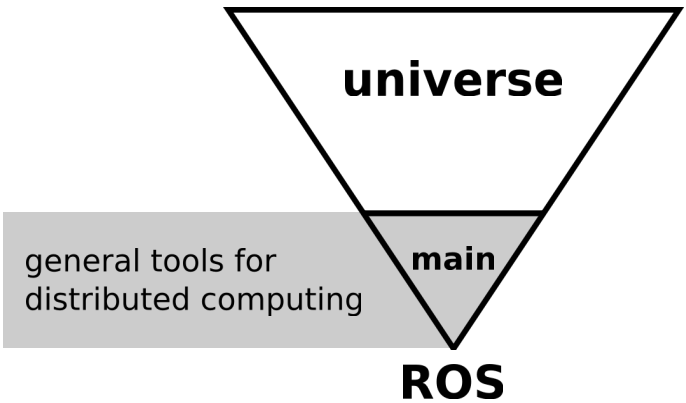
Robot Operating System [2]



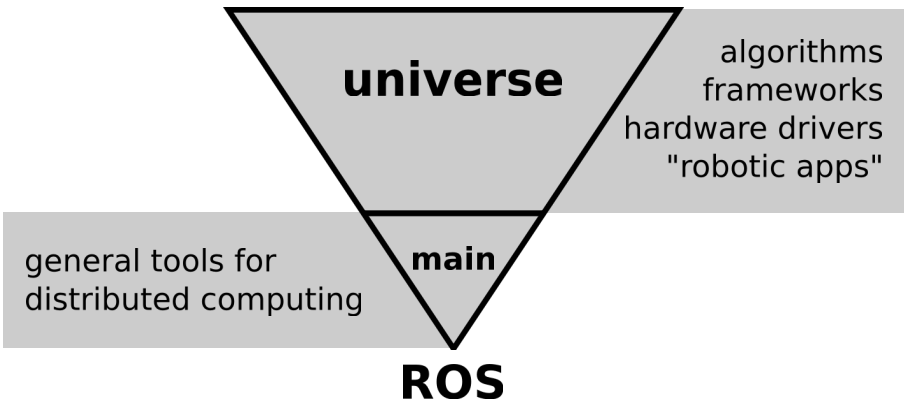
Robot Operating System [2]



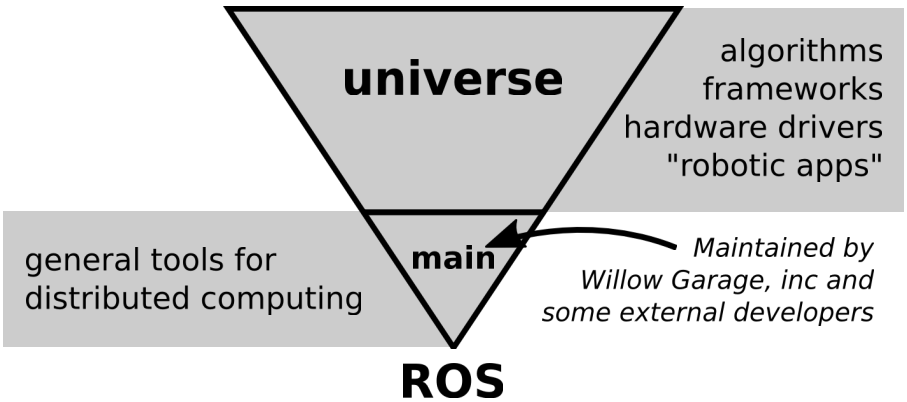
Robot Operating System [2]



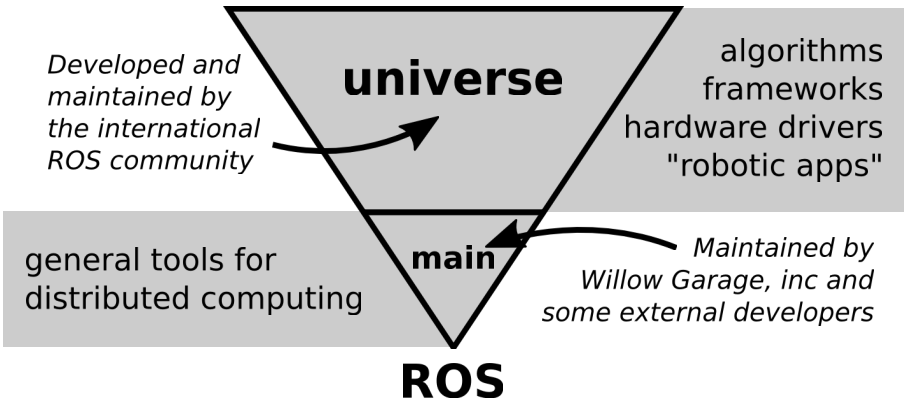
Robot Operating System [2]



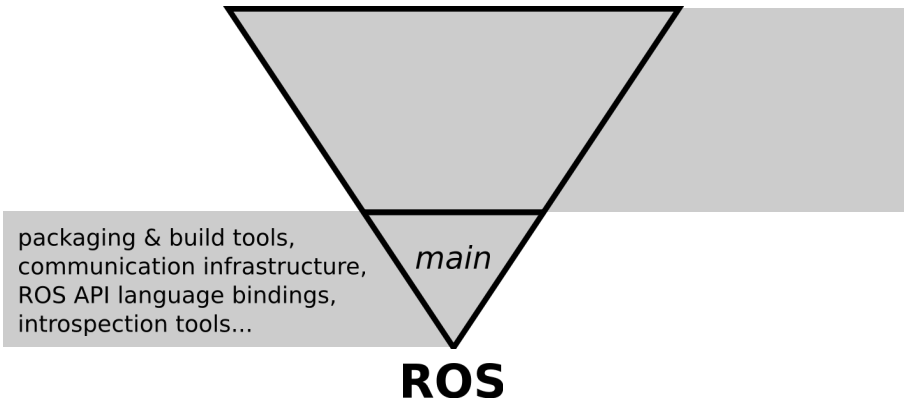
Robot Operating System [2]



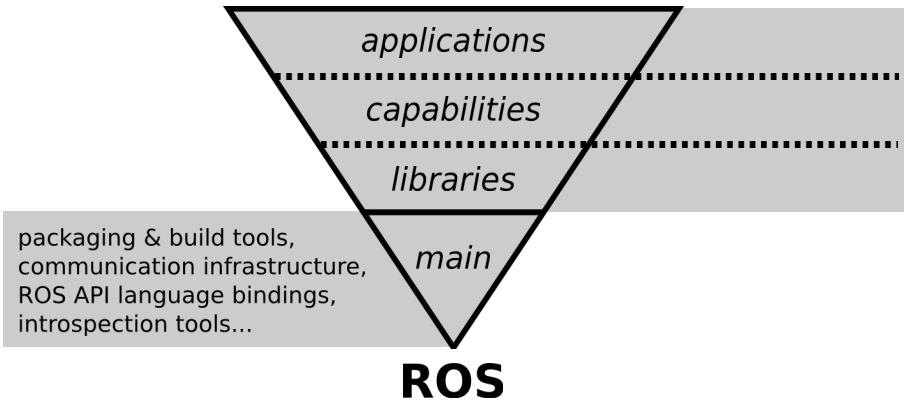
Robot Operating System [2]



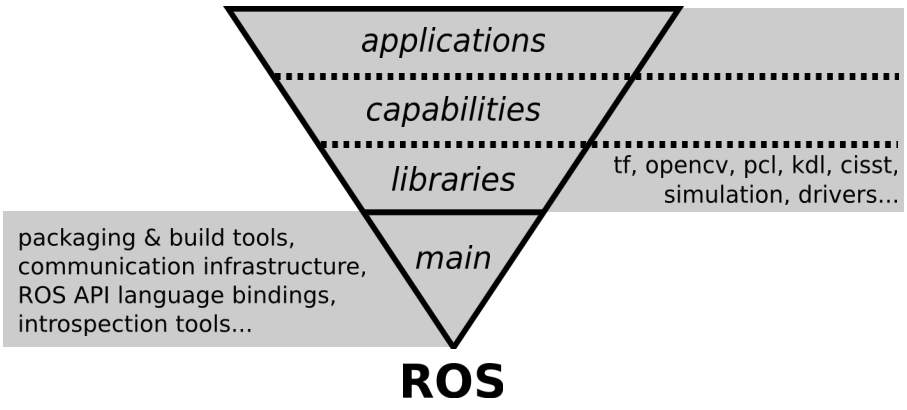
Robot Operating System [2]



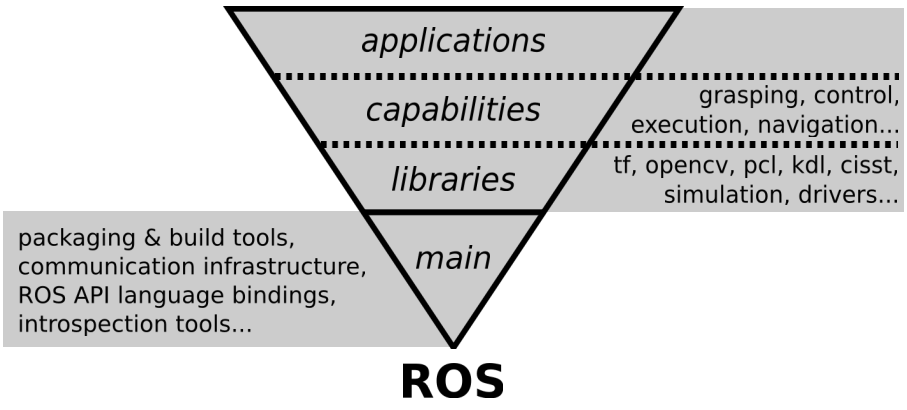
Robot Operating System [2]



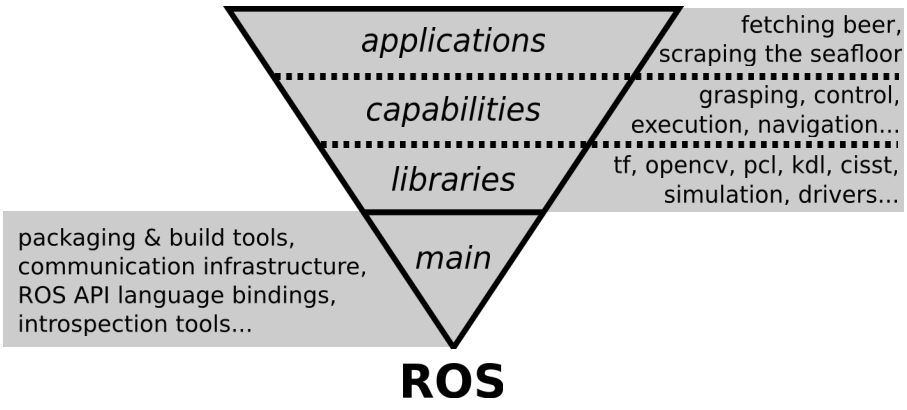
Robot Operating System [2]



Robot Operating System [2]



Robot Operating System [2]

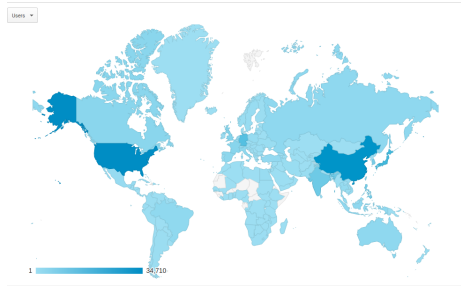


ROS Community

From the community report:

1.	United States	34,710 (19.08%)
2.	China	31,946 (17.56%)
3.	Japan	15,518 (8.53%)
4.	Germany	12,711 (6.99%)
5.	India	8,400 (4.62%)
6.	Philippines	7,235 (3.98%)
7.	South Korea	6,790 (3.73%)
8.	United Kingdom	4,325 (2.38%)
9.	Taiwan	4,233 (2.33%)
10.	France	3,725 (2.05%)
11.	Canada	3,354 (1.84%)
12.	Spain	2,955 (1.62%)
13.	Singapore	2,842 (1.56%)
14.	Italy	2,744 (1.51%)
15.	Russia	2,465 (1.35%)
16.	Indonesia	2,461 (1.35%)
17.	Australia	2,436 (1.34%)
18.	Brazil	2,231 (1.23%)
19.	Hong Kong	2,147 (1.18%)
20.	Turkey	1,928 (1.06%)
21.	Netherlands	1,511 (0.83%)
22.	Thailand	1,437 (0.79%)
23.	Poland	1,335 (0.73%)
24.	Switzerland	1,242 (0.68%)
25.	Vietnam	1,125 (0.62%)

wiki.ros.org visitor locations:

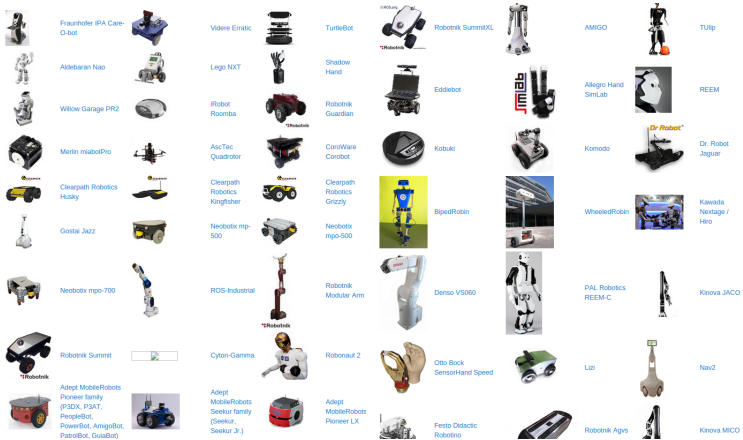


Source: Google Analytics
Site: wiki.ros.org in July 2018

6

ROS Community [2]

Some robots supporting ROS (data from November 2014):



ROS Build System

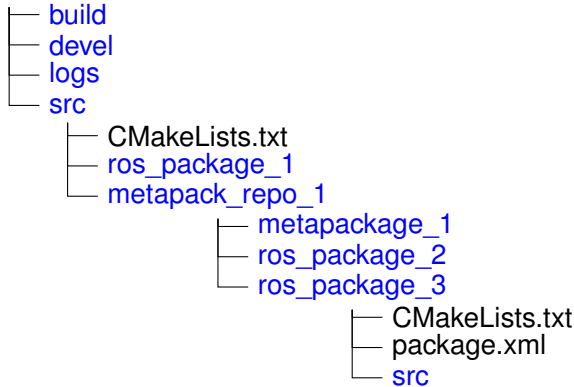
Packages and Metapackages

- *Packages* are a named collection of software that is built and treated as an atomic dependency in the ROS build system.
- *Metapackages* are dummy “virtual” packages that reference one or more related packages which are loosely grouped together

Similar to Debian packages.
Actually released through the Debian packaging system.

ROS Workspace

Workspaces have a specific structure



Managing Packages

Packages are stored in ROS workspaces:

```
$ roscd && cd ..  && pwd # this workspace is located at /home/jovyan/workspace/ros
```

In the Docker Container

- Creating a package:

  ```
  $ cd src
  $ catkin_create_pkg beginner_tutorials std_msgs rospy
  ```

- Compiling a package:

  ```
  $ cd ..  && catkin build
  ```

- Update ROS filesystem for new package:

  ```
  $ source devel/setup.bash
  ```

- Moving through ROS workspaces:

  ```
  $ roscd beginner_tutorial
  ```

Naming convention: underscores (no CamelCase, no-dashes)!

All the packages in your workspace are one huge CMake project.

Package.xml

beginner_tutorial/package.xml

```
1  <?xml version="1.0"?>
2  <package format="2">
3    <name>beginner_tutorial</name>
4    <version>0.0.0</version>
5    <description>The beginner_tutorial package</description>
6    <maintainer email="aniedz@cs.uni-bremen.de">Arthur</maintainer>
7    <license>Public domain</license>
8    <buildtool_depend>catkin</buildtool_depend>
9    <build_depend>rospy</build_depend>
10   <build_depend>std_msgs</build_depend>
11   <build_export_depend>rospy</build_export_depend>
12   <build_export_depend>std_msgs</build_export_depend>
13   <exec_depend>rospy</exec_depend>
14   <exec_depend>std_msgs</exec_depend>
15 </package>
```

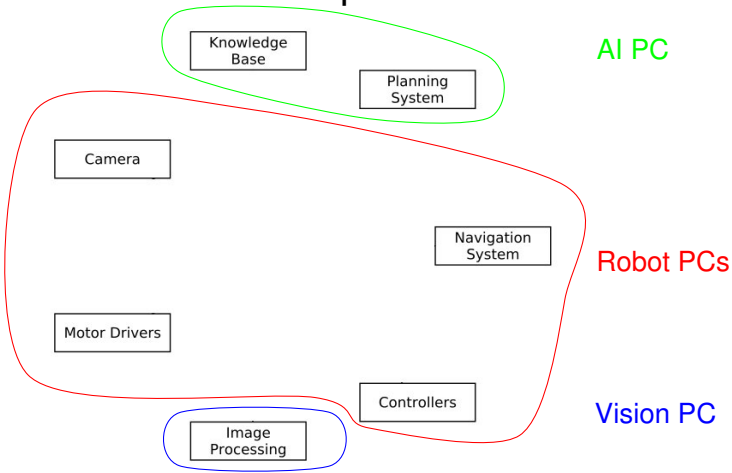
CMakeLists

beginner_tutorial/CMakeLists.txt

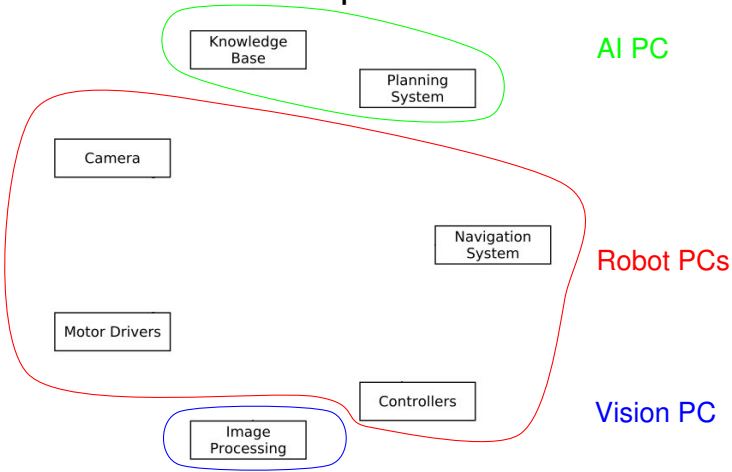
```
1  cmake_minimum_required(VERSION 3.0.2)
2  project(beginner_tutorial)
3
4  find_package(catkin REQUIRED COMPONENTS
5    rospy
6    std_msgs
7  )
8
9  catkin_package(
10   CATKIN_DEPENDS rospy std_msgs
11 )
12
13 include_directories(
14   ${catkin_INCLUDE_DIRS}
15 )
```

ROS Communication Layer

Robotic software components

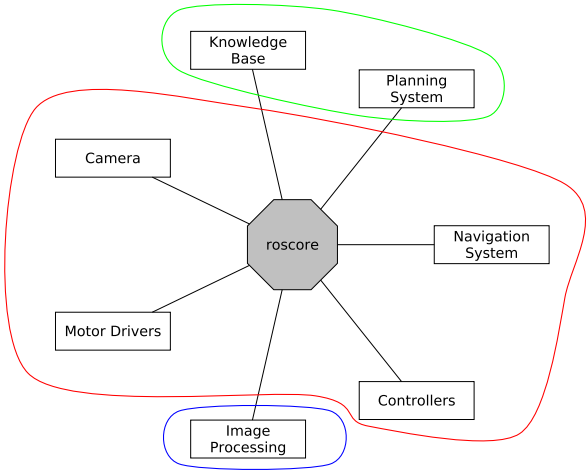


Robotic software components

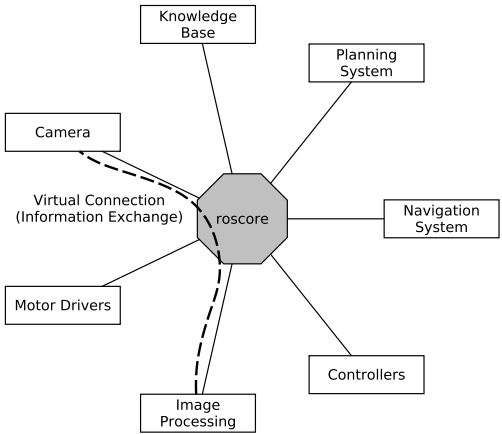


→ Processes distributed all over the place.

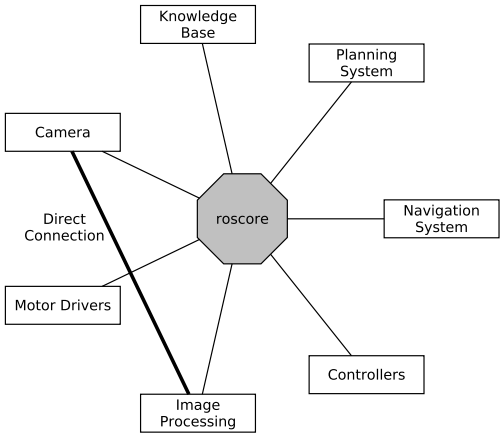
Connecting Pieces Together



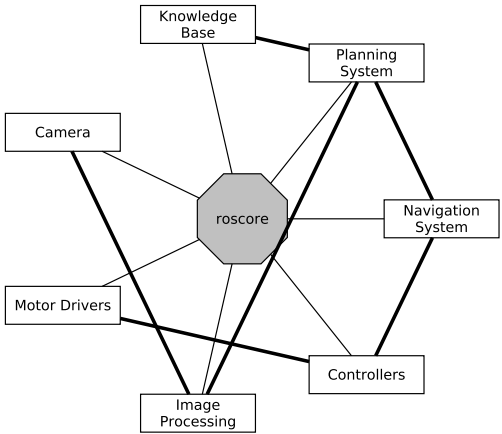
Connecting Pieces Together [2]



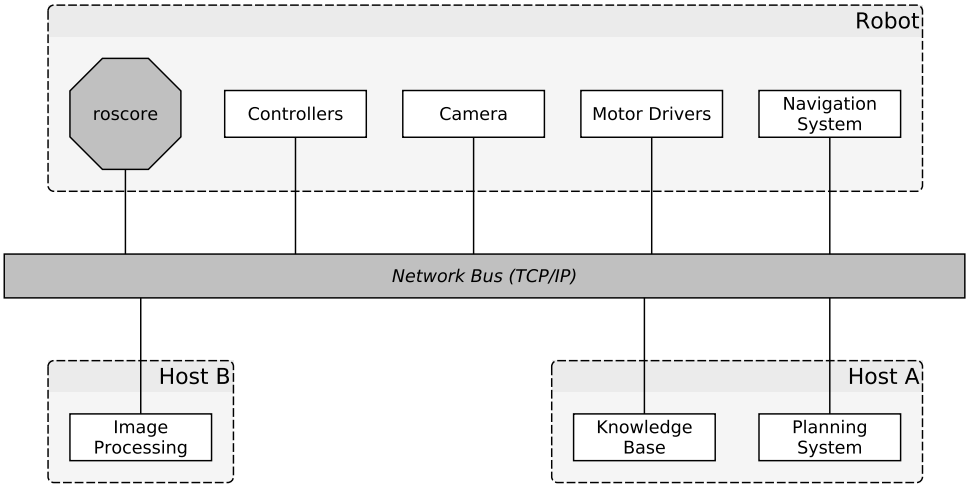
Connecting Pieces Together [2]



Connecting Pieces Together [2]



Distributed Hosts



roscore

- ROS master
  - A centralized XML-RPC server
  - Negotiates communication connections
  - Registers and looks up names of participant components
- Parameter Server
  - Stores persistent configuration parameters and other arbitrary data
- rosout
  - Distributed stdout

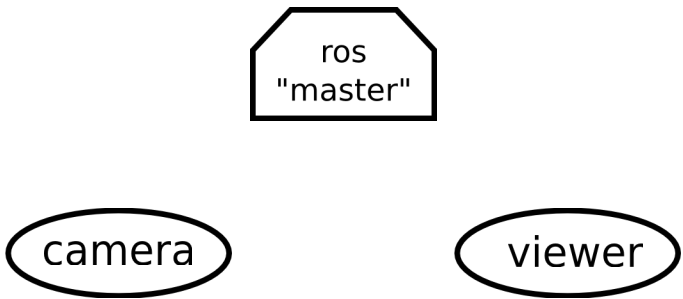
Terminology

- **Nodes** are processes that produce and consume data
- **Parameters** are persistent data stored on parameter server, e.g. configuration and initialization settings

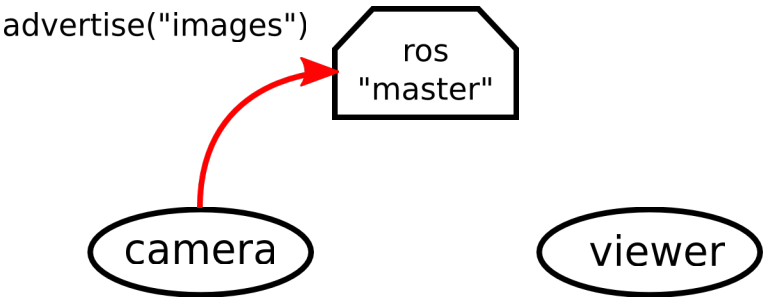
Node communication means:

- **Topics**: asynchronous many-to-many “streams-like”
  - Strongly-typed (ROS .msg spec)
  - Can have one or more *publishers*
  - Can have one or more *subscribers*
- **Services**: synchronous blocking one-to-many “function-call-like”
  - Strongly-typed (ROS .srv spec)
  - Can have only one *server*
  - Can have one or more *clients*
- **Actions**: asynchronous non-blocking one-to-many “function-call-like”
  - Built on top of topics but can be canceled

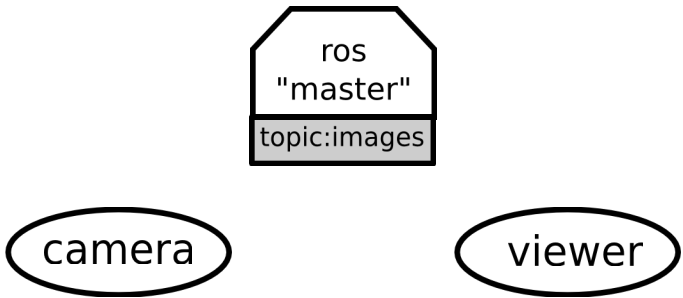
Establishing Communication



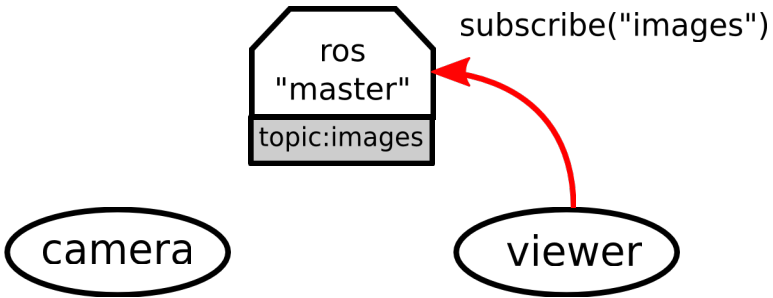
Establishing Communication



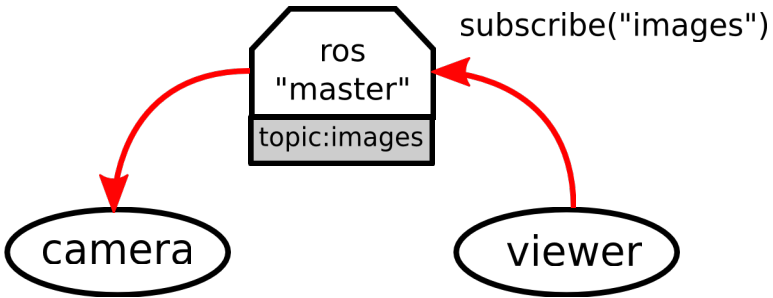
Establishing Communication



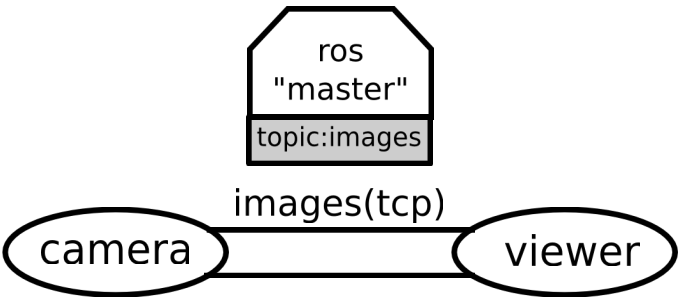
Establishing Communication



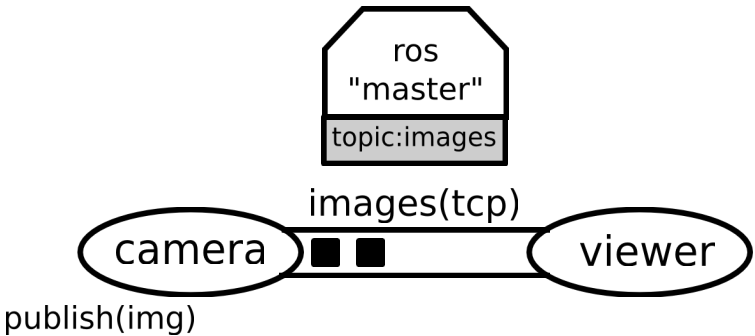
Establishing Communication



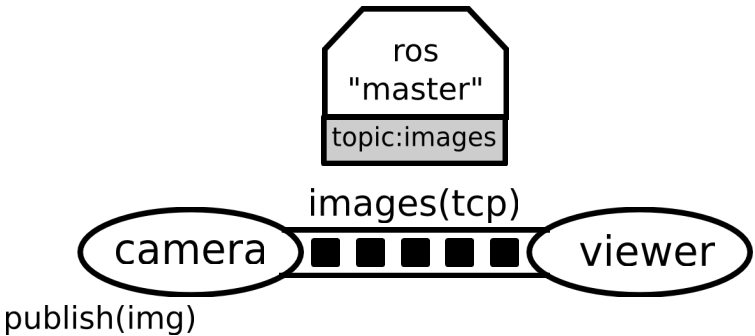
Establishing Communication



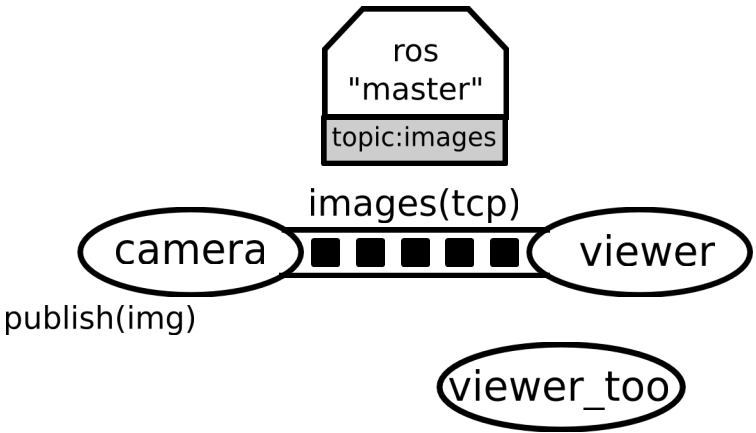
Establishing Communication



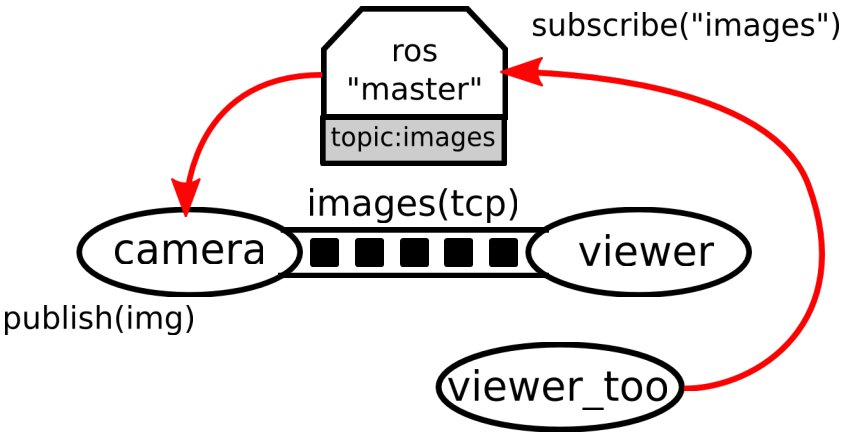
Establishing Communication



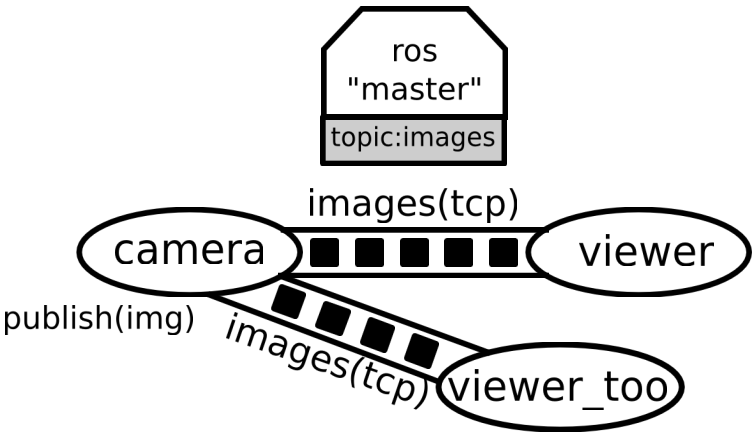
Establishing Communication



Establishing Communication



Establishing Communication



Tools

- `rosnode`: gives the user information about a node

```
$ rosnode -h

cleanup, info, kill, list, machine, ping
```

- `rostopic`: gives publishers, subscribes to the topic, datarate, the actual data

```
bw, echo, find, hz, info, list, pub, type
```

- `rosservice`: enables a user to call a ROS Service from the command line

```
call, find, list, type, uri
```

- `rosmsg`: gives information about message types

```
list, md5, package, packages, show
```

- `rossrv`: same as above for service types

```
list, md5, package, packages, show
```

- `roswtf`: diagnoses problems with a ROS network

ROS Graph

- Starting the core:

```
$ roscore
```

- Start the turtle simulation:

```
$ rosrun turtlesim turtlesim_node
```

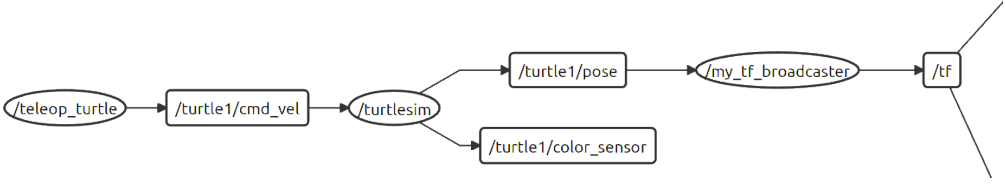
- Start teleoperation:

```
$ rosrun turtlesim turtle_teleop_key
```

- Publish turtle1 position to /tf:

```
$ rosrun turtle_tf turtle_tf_broadcaster turtle1
```

- Examining the ROS Graph in Jupyter



Launch Files

XML files for launching nodes:

- automatically set parameters and start nodes with a single file
- hierarchically compose collections of launch files
- automatically re-spawn nodes if they crash
- change node names, namespaces, topics, and other resource names
- without recompiling
- easily distribute nodes across multiple machines

Launch Files [2]

beginner_tutorial/turtle.launch

```
1  <launch>
2    <!-- Starting nodes-->
3    <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
4    <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
5
6    <node pkg="turtle_tf" type="turtle_tf_broadcaster.py"
7          name="turtle1_tf_broadcaster" respawn="false" output="screen" >
8      <!-- Specify startup parameters -->
9      <param name="turtle" type="string" value="turtle1" />
10   </node>
11 </launch>
```

Using the launch file:

```
$ roslaunch beginner_tutorial turtle.launch
```

ROS API

ROS API provides the programmer with means to

- start ROS node processes
- generate messages
- publish and subscribe to topics
- start service servers
- send service requests
- provide and query action services
- find ROS packages
- ...

ROS APIs: `roscpp`, `rospy`, `rosjava`, `rosjs`, `roslisp`

Overview

Discord

Follow this to our Discord server. Link open until 08.11.23

Assignment and dates

- Assignment 3:
  `https://github.com/artnie/rpwr-assignments`
- ROS Tutorials
  `http://wiki.ros.org/ROS/Tutorials`
- TF Tutorials
  `http://wiki.ros.org/tf/Tutorials`
- Grades: 8 points for this assignment
- Due: 08.11., 23:59 AM German time
- Next class: 09.11., 14:00

Evaluation

Thanks for your attention!

Special thanks to Lorenz Mösenlechner and Jan Winkler for providing illustrations!